12 REDUCING YOUR WORKLOAD WITH PHP INCLUDES

Wouldn't it be wonderful if you could make changes to just a single page and have them reflected through the site in the same way as CSS? Well, with PHP includes, you can. As the name suggests, the contents of an include file are included and treated as an integral part of the page. They can contain anything you would normally find in a PHP page: plain text, HTML, and PHP code. The file name extension doesn't even need to be `.php`, although for security it's common practice to use it.

Dreamweaver makes working with includes easy thanks to its ability to display the contents of an include in Design view (or Live view for dynamic content).

In this chapter, you'll learn about the following:

- Using PHP includes for common page elements
- Applying CSS to page fragments with design-time style sheets
- Exporting a navigation menu to an external file
- Adapting the mail processing script to work with other forms
- Avoiding the "headers already sent" error with includes

To start with, let's take a quick look at how you create a PHP include.

Including text and code from other files

The ability to include code from other files is a core part of PHP. All that's necessary is to use one of PHP's include commands and tell the server where to find the file.

Introducing the PHP include commands

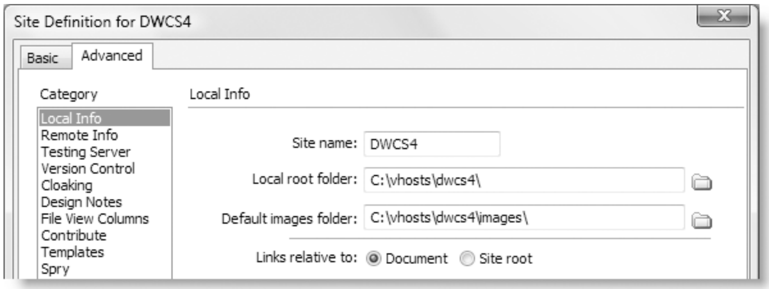
PHP has four separate commands for creating an include: `include()`, `include_once()`, `require()`, and `require_once()`. Why so many? And what's the difference?

They all do the same thing, but "require" is used in the sense of "mandatory"; everything comes to a grinding halt if the external file is missing or can't be opened. The "include" pair of commands, on the other hand, soldier bravely on. The purpose of `_once` is to prevent variables being accidentally overwritten and functions from being redefined (defining the same function more than once triggers a fatal error). The PHP engine uses the first instance it encounters and ignores any duplicates. If in doubt about which to use, choose `include_once()` or `require_once()`. Using them does no harm and could avoid problems.

Telling PHP where to find the external file

The include commands take a single argument: a string containing the path of the external file. While this sounds simple enough, it confuses many Dreamweaver users. PHP looks for the external file in what's known as the `include_path`. By default, this includes the current directory (folder), although some hosting companies configure PHP to restrict you to including files only from specified locations. In any case, PHP expects either a relative or an absolute path to an include file. *It won't work with a path relative to the site root*.

If Links relative to is set to Document in the Local Info category of your site definition (see Figure 12-1), Dreamweaver automatically uses the correct path for include files. However, if you have selected Site root as your default style for links, includes won't work unless you override the default setting to change the path to a document-relative one or take alternative measures to set the `include_path`.


Site Definition for DWCS4

Basic | Advanced

Category: Local Info, Remote Info, Testing Server, Version Control, Cloaking, Design Notes, File View Columns, Contribute, Templates, Spry

Local Info

Site name: DWCS4

Local root folder: C:\vhosts\dwcs4\

Default images folder: C:\vhosts\dwcs4\images\

Links relative to: (● Document) ○ Site root

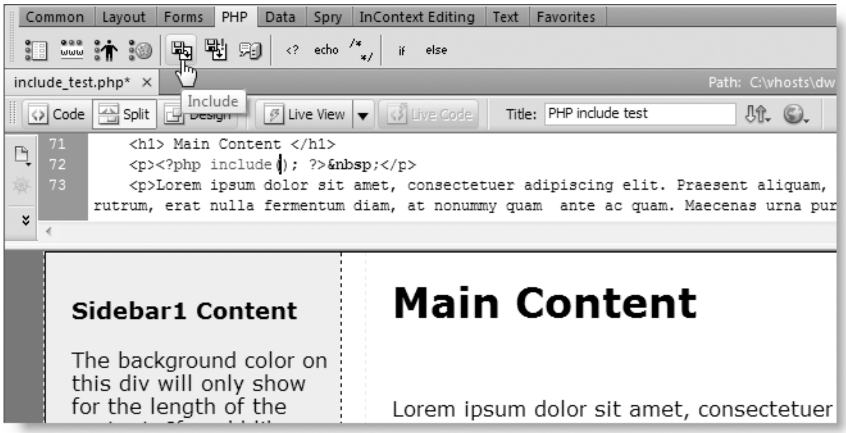
Figure 12-1. Dreamweaver's site definition dialog box lets you specify the default format of internal links.

A practical exercise should clarify the situation.

Including a text file

In this exercise, you'll see what happens if you use the wrong type of path for an include file. You'll also learn how to override the default setting so that you can use includes successfully even if your site definition specifies using links relative to the site root.

1. Create a new subfolder called `includes` in your `workfiles` folder, and copy `include.txt` from `examples/includes` to the new folder.
2. Go to File ➤ New. Select Blank Page and PHP for Page Type. Choose any of the predefined layouts. The one I chose was 2 column fixed, left sidebar. This is only going to be a test page, so you can leave Layout CSS on Add to Head. Click Create, and save the file as `include_test.php` in `workfiles/ch12`.
3. Position your cursor at the beginning of the first paragraph under the Main Content headline. Press Enter/Return to insert a new paragraph, and then press your up keyboard arrow to move the insertion point into the empty paragraph.
4. Select the PHP tab on the Insert bar, and click the Include button as shown in the following screenshot (alternatively use the menu option Insert ➤ PHP Objects ➤ Include). Dreamweaver opens Split view, inserts a PHP code block complete with an `include()` command, and positions the insertion point between the parentheses, ready for you to enter the path of the external file (it's on line 72 in the following screenshot).


Common Layout Forms PHP Data Spry InContext Editing Text Favorites
<? echo /* */ if else
include_test.php*
Path: C:\vhosts\dw
Include
Code Split Design Live View Live Code Title: PHP include test

```
<h1> Main Content </h1>
<p><?php include (); ?>&nbsp;</p>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam,
rutrum, erat nulla fermentum diam, at nonummy quam  ante ac quam. Maecenas urna pur
```

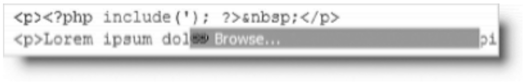
Sidebar1 Content

The background color on this div will only show for the length of the

Main Content

Lorem ipsum dolor sit amet, consectetuer

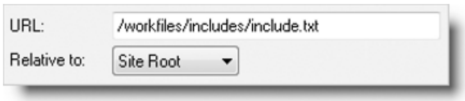
5. The path needs to be a string, so enter a quotation mark (I prefer a single quote, but it doesn't matter, as long as the closing quote matches). Dreamweaver's syntax coloring turns all the subsequent code red, but this reverts to normal once you have finished. Inserting the quotation mark places a tiny Browse icon at the insertion point like this:



```
<p><?php include('); ?>&nbsp;</p>
<p>Lorem ipsum dol
```

Bringing up the Browse *icon automatically is a small but welcome productivity improvement in Dreamweaver CS4. In previous versions, you needed to select* URL Browser *from* Code Hint Tools *on the context menu.*

6. Click the Browse icon to open the Select File dialog box. Navigate to the `workfiles/includes` folder, and select `include.txt`. Before clicking OK, check the setting of Relative to at the bottom of the dialog box. It displays Document or Site Root, depending on the default in your site definition (see Chapter 2 and Figure 12-1). If necessary, change it to Site Root, as shown here, and click OK:



URL: /workfiles/includes/include.txt

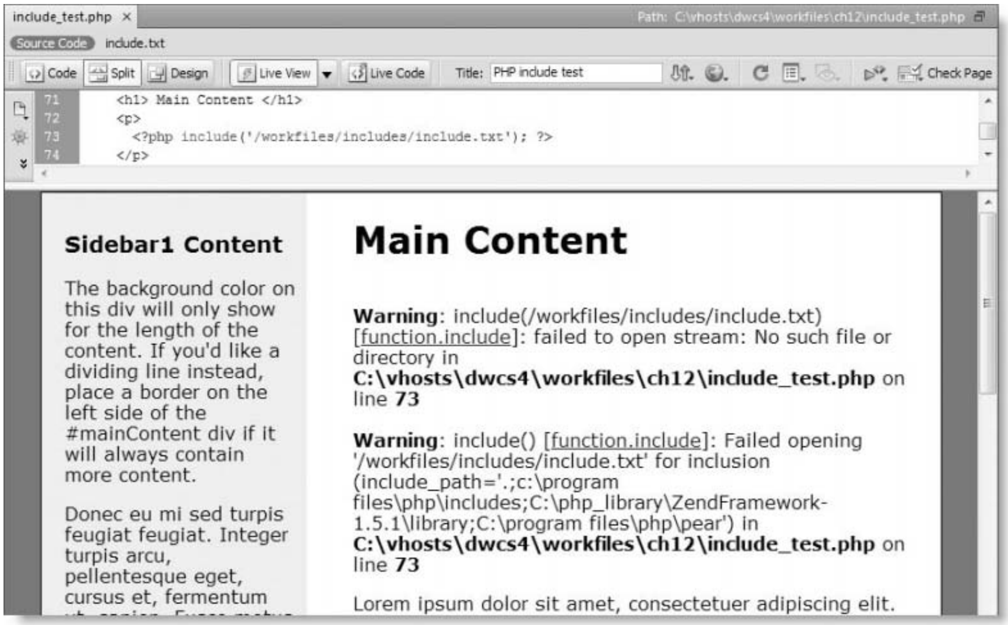
Relative to: Site Root

7. Type a closing quote after the path that has just been entered into the `include()` command. Syntax coloring turns the rest of the code back to its normal color—a useful reminder of the importance of matching quotes. Move your cursor further along the line to remove the `&nbsp;` just before the closing `</p>` tag.

The code in that line should now look like this:

```
<p><?php include('/workfiles/includes/include.txt'); ?></p>
```

8. Click inside Design view. The content of the external text file should be displayed just below the main heading. Magic . . . well, not quite.
9. Click the Live View button. Dreamweaver will ask whether you want to save the page and update it on the testing server. Click Yes in both cases. You should see something like Figure 12-2.



```
71      <h1> Main Content </h1>
72      <p>
73        <?php include('/workfiles/includes/include.txt'); ?>
74      </p>
```

Sidebar1 Content

The background color on this div will only show for the length of the content. If you'd like a dividing line instead, place a border on the left side of the #mainContent div if it will always contain more content.

Donec eu mi sed turpis feugiat feugiat. Integer turpis arcu, pellentesque eget, cursus et, fermentum

Main Content

Warning: include(/workfiles/includes/include.txt) [function.include]: failed to open stream: No such file or directory in **C:\vhosts\dwcs4\workfiles\ch12\include_test.php** on line **73**

Warning: include() [function.include]: Failed opening '/workfiles/includes/include.txt' for inclusion (include_path='.;c:\program files\php\includes;C:\php_library\ZendFramework-1.5.1\library;C:\program files\php\pear') in **C:\vhosts\dwcs4\workfiles\ch12\include_test.php** on line **73**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

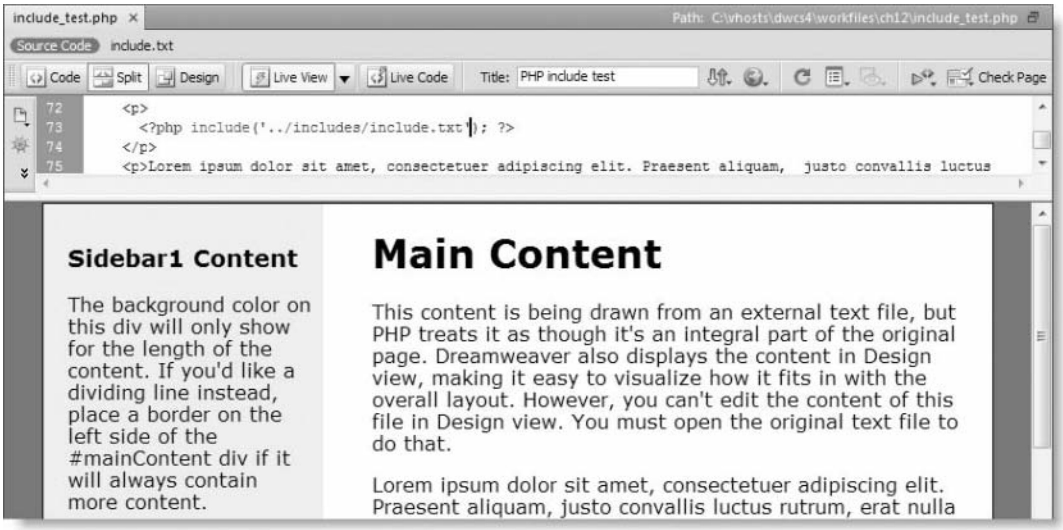
Figure 12-2. If PHP can't find the include file, it displays ugly warning messages.

The first warning says there was no such file or directory, but of course, there is. The second warning gives a cryptic clue as to why PHP can't open the file. The `include_path` is where PHP looks for include files. The value shown on your computer for `include_path` won't necessarily be the same as in Figure 12-2; the default on most web servers is **.** (a period), which is shorthand for the current working directory, and either the main PHP folder or `pear` (PEAR—the PHP Extension and Application Repository—is a library of extensions to PHP).

What these warnings are telling you is that PHP doesn't understand a leading forward slash as meaning the site root, so it starts from the current folder and ends up in a nonexistent part of the site.

10. Switch off Live view, and delete the value between the parentheses of `include()`.

11. Repeat steps 5 and 6 to `include.txt` again, but this time make sure that Relative to is set to Document. Switch on Live view again, saving and updating the files when prompted. The content of the include file should now be correctly displayed, as shown in Figure 12-3.



```
<p>
  <?php include('../includes/include.txt'); ?>
</p>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam,  justo convallis luctus
```

Sidebar1 Content

The background color on this div will only show for the length of the content. If you'd like a dividing line instead, place a border on the left side of the #mainContent div if it will always contain more content.

Main Content

This content is being drawn from an external text file, but PHP treats it as though it's an integral part of the original page. Dreamweaver also displays the content in Design view, making it easy to visualize how it fits in with the overall layout. However, you can't edit the content of this file in Design view. You must open the original text file to do that.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam, justo convallis luctus rutrum, erat nulla

Figure 12-3. The include file is displayed correctly when a relative path is used.

12. Switch off Live view, and change the command from `include` to `require` like this:

```
<?php require('../includes/include/txt'); ?>
```

13. Test the page again in Live view. It should look identical to Figure 12-3.

14. Change the path to point to a nonexistent file, such as `includ.txt`. When you test the page in Live view, it should look similar to Figure 12-2, but instead of the second warning, you should see Fatal error. The other difference is that there's no text after the error message. As explained in "Understanding PHP error messages" in Chapter 10, any output preceding a fatal error is displayed, but once the error is encountered, everything else is killed stone dead.

Using site-root-relative links with includes

As you have just seen, PHP cannot find include files referenced by a link relative to the site root. My recommendation is that, if you have selected links relative to the site root as your default, you simply select Relative to Document in the Select File dialog box (as described in step 11 of the preceding exercise) when creating an include.

Nevertheless, there are a couple of alternatives if you have a pressing reason for wanting to use links relative to the site root. The problem is that they don't work on all servers.

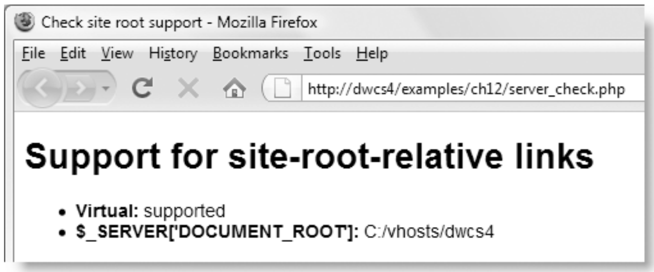
The virtual() function accepts both document-relative and site-root-relative paths and can be used as a direct replacement for include() and require(). It works only when PHP is run as an Apache module.

$_SERVER['DOCUMENT_ROOT'] is a predefined PHP variable that contains the path of the server's root folder, so adding it to the beginning of a site-root-relative link has the effect of turning it into an absolute path. The following works on most servers:

```
<?php include($_SERVER['DOCUMENT_ROOT'].'/workfiles/includes/ ➥
include/txt'); ?>
```

Unfortunately, $_SERVER['DOCUMENT_ROOT'] isn't supported by IIS running PHP in CGI mode.

To check whether your server supports either method, run server_check.php in examples/ch12. If both are supported, you should see output similar to this:


Check site root support - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://dwcs4/examples/ch12/server_check.php

Support for site-root-relative links

- **Virtual:** supported
- **$_SERVER['DOCUMENT_ROOT']:** C:/vhosts/dwcs4

The preceding screenshot shows the output from my local testing server, but it's important to test your remote server as well. If neither virtual() nor $_SERVER['DOCUMENT_ROOT'] is supported and you still want to use site-root-relative links, you need to define a **constant** containing the path to the site root. A constant is like a variable, except that once defined in a script, its value cannot be changed. Constants don't begin with a dollar sign, and by convention, they are always in uppercase. You define a constant like this:

```
define('SITE_ROOT', 'C:\inetpub\wwwroot\dwcs4');
```

You could then use SITE_ROOT with a site-root-relative link like this:

```
<?php include(SITE_ROOT.'/workfiles/includes/include/txt'); ?>
```

The disadvantage with this approach is that you need to include the definition of the constant in every page that uses includes.

The restriction on site-root-relative links applies only to the include command. Inside include files, all links should be site-root-relative. Document-relative links inside an include file will be broken if the file is included at a different level of the site hierarchy.

Lightening your workload with includes

So far, you have seen only a fairly trivial use of an include to insert a block of text inside a paragraph. This might be useful in a situation where you want to change the content of part of a page on a frequent basis without going to the bother of building a database-driven content management system. A much more practical use of includes is for content that appears on many pages, for example a navigation menu or footer. Any changes made to the include file are immediately reflected throughout the site.

Choosing the right file name extension for include files

As I explained at the beginning of the chapter, the external file doesn't need to have a .php file name extension. Many developers use .inc as the default file name extension to make it clear that the file is an include. Although this a common convention, Dreamweaver doesn't automatically recognize an .inc file as containing PHP code, so you don't get code hints or syntax coloring. More importantly, browsers don't understand .inc files. So, if anybody accesses an .inc file directly through a browser (as opposed to it being included as part of a PHP page), everything is displayed as plain text.

This is a potential security risk if you put passwords or other sensitive information in external files. One way around this problem is to store include files outside the server root folder. Many hosting companies provide you with a private folder, which cannot be reached by a browser. As long as the PHP script knows where to find the external file *and* has permission to access it, include files can be outside the server root. However, this creates problems for Dreamweaver site management.

A simpler, widely adopted solution is to use .inc.php as the file name extension. Browsers and servers treat only the final .php as the file name extension and automatically pass the file to the PHP engine if requested directly. The .inc is simply a reminder to you as the developer that this is an include file.

As long as you store passwords and other sensitive information as PHP variables within PHP code blocks and you use .php as the final file name extension, your data cannot be seen by anyone accessing the page directly in a browser (of course, it will be revealed if your code uses echo or print to display that information, but I assume you have the sense not to do that).

Displaying HTML output

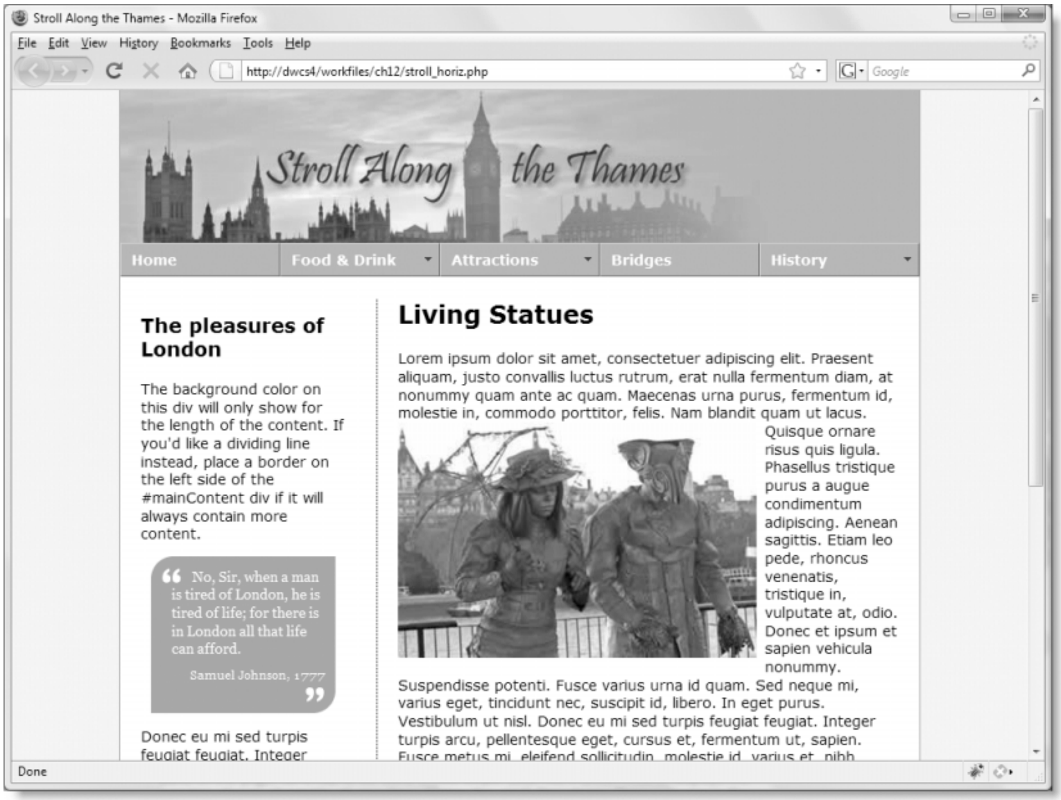
When PHP includes an external file, it automatically treats the contents of the external file as plain text or HTML. This means that you can cut a section out of an existing page built in HTML and convert it into an include file. In order to preserve your sanity, it's important to put only complete, logical elements in external files. Putting the opening part of a <div> in one external file and the closing part in another file is a disaster waiting to happen. It becomes impossible to keep track of opening and closing tags, and Dreamweaver is likely to start trying to replace what it regards as missing tags.

Usually, I find the best approach is to build the complete page first and then convert common elements into include files.

Converting a navigation menu into an include

This exercise shows you how to extract the menu from the “Stroll along the Thames” site in Chapter 6 and convert it into an include file. The menu bar currently contains only dummy links. I’ll show you later in this chapter how to edit the menu to update the links.

1. Copy `stroll_horiz.php` and `stroll.css` from `examples/ch12` to `workfiles/ch12`. This is an identical copy of the completed page from Chapter 6. The only difference is that the file name extension has been changed to `.php` so that the PHP engine knows to process it and include the external files you are about to create. Test the page in a browser to make sure it displays correctly. It should look like Figure 12-4.



Stroll Along the Thames - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://dwcs4/workfiles/ch12/stroll_horiz.php
Google
Stroll Along the Thames
Home
Food & Drink
Attractions
Bridges
History
Done

The pleasures of London

The background color on this div will only show for the length of the content. If you'd like a dividing line instead, place a border on the left side of the #mainContent div if it will always contain more content.

No, Sir, when a man is tired of London, he is tired of life; for there is in London all that life can afford.

Samuel Johnson, 1777

Donec eu mi sed turpis feugiat feugiat. Integer

Living Statues

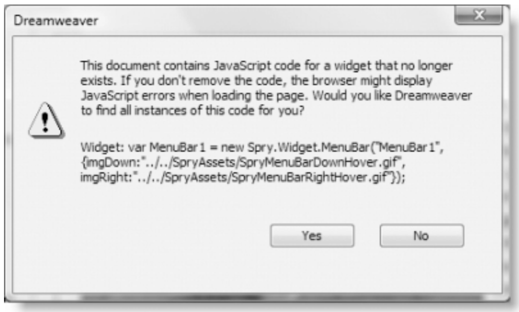
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam, justo convallis luctus rutrum, erat nulla fermentum diam, at nonummy quam ante ac quam. Maecenas urna purus, fermentum id, molestie in, commodo porttitor, felis. Nam blandit quam ut lacus. Quisque ornare risus quis ligula. Phasellus tristique purus a augue condimentum adipiscing. Aenean sagittis. Etiam leo pede, rhoncus venenatis, tristique in, vulputate at, odio. Donec et ipsum et sapien vehicula nonummy. Suspendisse potenti. Fusce varius urna id quam. Sed neque mi, varius eget, tincidunt nec, suscipit id, libero. In eget purus. Vestibulum ut nisl. Donec eu mi sed turpis feugiat feugiat. Integer turpis arcu, pellentesque eget, cursus et, fermentum ut, sapien. Fusce metus mi, eleifend sollicitudin, molestie id, varius et, nibh.

Figure 12-4. The menu is the same on every page of the site, so it is a prime candidate for an include file.

2. Create a new PHP file, and save it in the `workfiles/includes` folder as `menu.inc.php`. You don’t need one of the CSS layouts, because you need a completely blank page. Switch to Code view in `menu.inc.php`, and delete everything, including the `DOCTYPE` declaration. There should be nothing left in the page.

12

3. Switch to `stroll_horiz.php` in the Document window. Click anywhere inside the navigation menu, and click <div#nav> in the Tag selector to select the entire menu. Switch to Code view, and then cut the menu to your computer clipboard (Ctrl+X/Cmd+X or Edit ➤ Cut).



This document contains JavaScript code for a widget that no longer exists. If you don't remove the code, the browser might display JavaScript errors when loading the page. Would you like Dreamweaver to find all instances of this code for you?

Widget: var MenuBar1 = new Spry.Widget.MenuBar("MenuBar1", {imgDown:"../../SpryAssets/SpryMenuBarDownHover.gif", imgRight:"../../SpryAssets/SpryMenuBarRightHover.gif"});

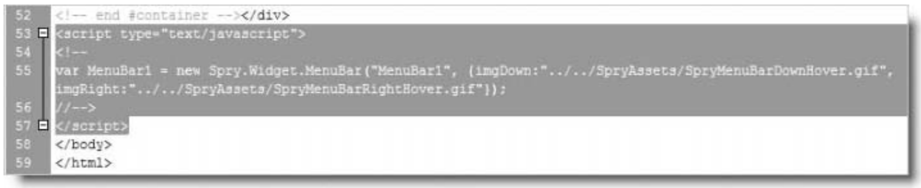
You must be in Code view when cutting the menu to the clipboard. If you remain in Design view, Dreamweaver cuts all the Spry-related code and pastes it into the include file. You want to move only the HTML code and the Spry object initialization, but they're in different parts of the page, so it has to be done in two steps. Click No if Dreamweaver displays the warning shown in the preceding screenshot at any time during the next few steps. Once you move the initialization script, the warning message no longer appears.

4. Without moving the insertion point, click the Include button on the PHP tab of the Insert bar (or use the menu alternative). This inserts a PHP code block and positions your cursor between the parentheses of an `include()` command.

5. Type a single quote, and click the Browse icon to navigate to `menu.inc.php` in the `workfiles/includes` folder in the same way as in "Including a text file" earlier in the chapter. In the Select File dialog box, make sure Relative to is set to Document. Click OK, and type a closing quote after the path. Save `stroll_horiz.php`.

6. Switch to `menu.inc.php` in the Document window. Make sure you are in Code view, and paste the menu that you cut from `stroll_horiz.php`. (If you are in Design view, you won't get all the HTML code. Always cut and paste in the same view in Dreamweaver—Design view to Design view or Code view to Code view.)

7. Go back to `stroll_horiz.php`, scroll down to the bottom of the page, and cut to your clipboard the section of code highlighted on lines 53–57 in the following screenshot:



```
52  <!-- end #container --></div>
53  <script type="text/javascript">
54  <!--
55  var MenuBar1 = new Spry.Widget.MenuBar("MenuBar1", {imgDown:"../../SpryAssets/SpryMenuBarDownHover.gif",
    imgRight:"../../SpryAssets/SpryMenuBarRightHover.gif"});
56  //-->
57  </script>
58  </body>
59  </html>
```

This is the initialization script for the Spry menu bar. Make sure you have the opening and closing `<script>` tags.

8. Paste the Spry object initialization script into `menu.inc.php` after the closing `</div>` tag. Save `menu.inc.php`, and close the file.
9. Switch to Design view in `stroll_horiz.php`. The menu should be visible as it was before. If you can't see the menu, open Preferences from the Edit menu (Dreamweaver menu on a Mac), select the Invisible Elements category, and make sure there's a check mark in Server-Side includes: Show contents of included file.
10. Hover your mouse pointer over the navigation menu, and click the turquoise Spry Menu Bar tab at the top-left corner. The Property inspector recognizes it as a server-side include (SSI) and displays the name of the file, together with an Edit button, as shown in Figure 12-5. If the Spry Menu Bar tab doesn't appear, make sure there's a check mark alongside Invisible Elements in the View ➤ Visual Aids submenu.

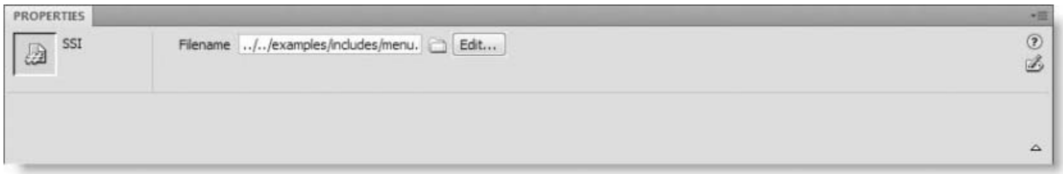


Figure 12-5. The Property inspector provides a direct link to edit the include file.

If the Related Files feature is enabled, clicking the Edit button in the Property inspector opens the include file in Split view ready for editing. Frequently, though, it's better to open the include file as a separate file so that you can edit it in Design view as well as in Code view. To do so, right-click the include file's name in the Related Files toolbar, and select Open as Separate File.

If you have disabled Related Files, clicking Edit opens the file in a separate tab.

11. Test `stroll_horiz.php` in a browser. It should look like Figure 12-4, and the menu should work as before. You can check your code against `stroll_horiz_menu.php` in `examples/ch12` and `menu.inc.php` in `examples/includes`.

An annoying quirk in the way Dreamweaver handles PHP includes in Design view is that the include command must be in its own PHP code block. If you put any other PHP code in the same block—even a comment—Dreamweaver just displays the gold PHP shield.

Putting the Spry object initialization script at the end of `menu.inc.php` results in it being called earlier than it was in the original page, but it's still in the right order and doesn't result in invalid code. It also prevents the warning in step 3 from being displayed every time you open the parent page.

An added advantage is that you can edit the Spry menu through the Property inspector in the same way as in Chapter 6. Even though the include file has no direct link to the Spry menu bar external JavaScript file, Dreamweaver automatically finds it because the Spry assets folder is specified in the site definition.

However, what you put in an external file doesn't always have such benign consequences.

Avoiding problems with include files

The server incorporates the content of an include file into the page at the point of the include command. If you pasted all the Spry-related code into `menu.inc.php`, rather than just the constructor, you would end up with the link to the external style sheet within the `<body>` of `stroll_horiz.php`. Although some browsers might render the page correctly, `<style>` blocks are invalid outside the `<head>` of a web page. If it doesn't break now, it probably will sooner or later as browsers get increasingly standards-compliant.

The most common mistake with include files is adding duplicate `<head>` and `<body>` tags. Keep your include files free of extraneous code, and make sure when everything fits back together that you have a `DOCTYPE` declaration, a single `<head>` and `<body>`, and that everything is in the right order.

Dreamweaver depends on the `DOCTYPE` declaration at the top of a page to determine whether to use XHTML rules. Code added to an include will normally use HTML style, so when editing an include, you need to keep a close eye on what is happening in Code view. This is why I recommend extracting code into include files only toward the end of a project or if the external file uses mainly dynamic code.

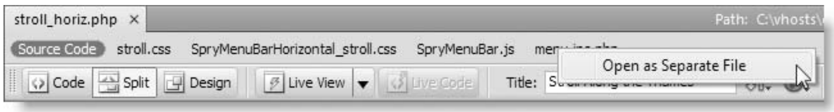
Another common problem is a broken link in an include file. Always use links relative to the site root inside include files. As explained in Chapter 2, root-relative links provide a constant reference to a page or an asset, such as an image. If you use document-relative links inside an include file, the relationship—and therefore the link—is broken if the file is included at a different level of the site hierarchy than where it was originally designed.

Let's test that with the menu bar that was extracted to an external file in the preceding exercise.

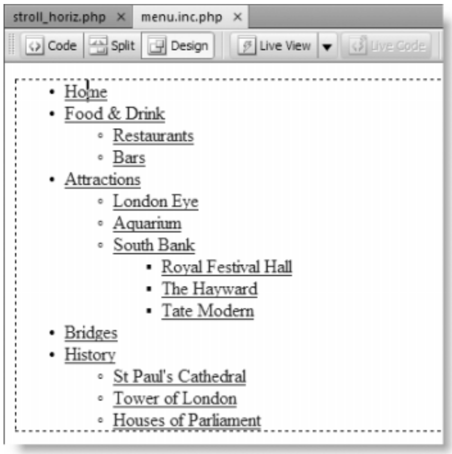
Updating links in the menu bar

This exercise demonstrates the importance of using root-relative links in an include file. It updates two of the links in `menu.inc.php`, the include file created in the preceding exercise. Continue working with the same files.

1. With `stroll_horiz.php` open in the Document window, right-click `menu.inc.php` in the Related Files toolbar, and select Open as Separate File, as shown here:



2. This opens `menu.inc.php` in a separate tab. The menu bar displays as an unstyled unordered list (see Figure 12-6). In the next section, I'll show you how you can display the styles while working on an include file like this. However, that's not important at the moment. What you're interested in is updating the links.

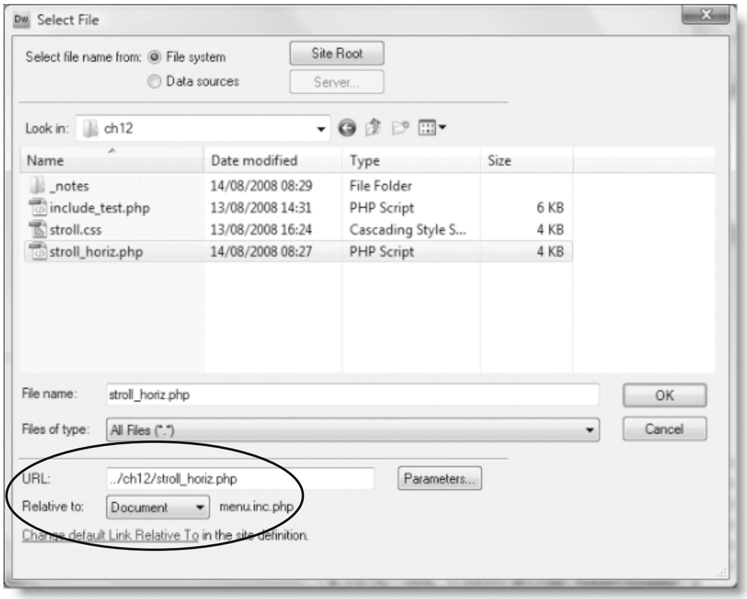


- Home
- Food & Drink
  - Restaurants
  - Bars
- Attractions
  - London Eye
  - Aquarium
  - South Bank
    - Royal Festival Hall
    - The Hayward
    - Tate Modern
- Bridges
- History
  - St Paul's Cathedral
  - Tower of London
  - Houses of Parliament

Figure 12-6. The menu bar is unstyled in the include file.

If you prefer working with the Spry menu bar Property inspector, select the Spry Menu Bar turquoise tab at the top left of the unordered list. However, I think it's quicker to just click inside the individual links and use the HTML view of the Property inspector.

Whichever method you use, select the Home link, click the Browse for File icon to the right of the Link field in the Property inspector, and navigate to `stroll_horiz.php` in the `workfiles/ch12` folder. Make sure Relative to in the Select File dialog box is set to Document, as shown in the following screenshot (I'm deliberately doing this to demonstrate what happens when you use document-relative links in an include file):



Name	Date modified	Type	Size
_notes	14/08/2008 08:29	File Folder	
include_test.php	13/08/2008 14:31	PHP Script	6 KB
stroll.css	13/08/2008 16:24	Cascading Style S...	4 KB
stroll_horiz.php	14/08/2008 08:27	PHP Script	4 KB

File name: stroll_horiz.php

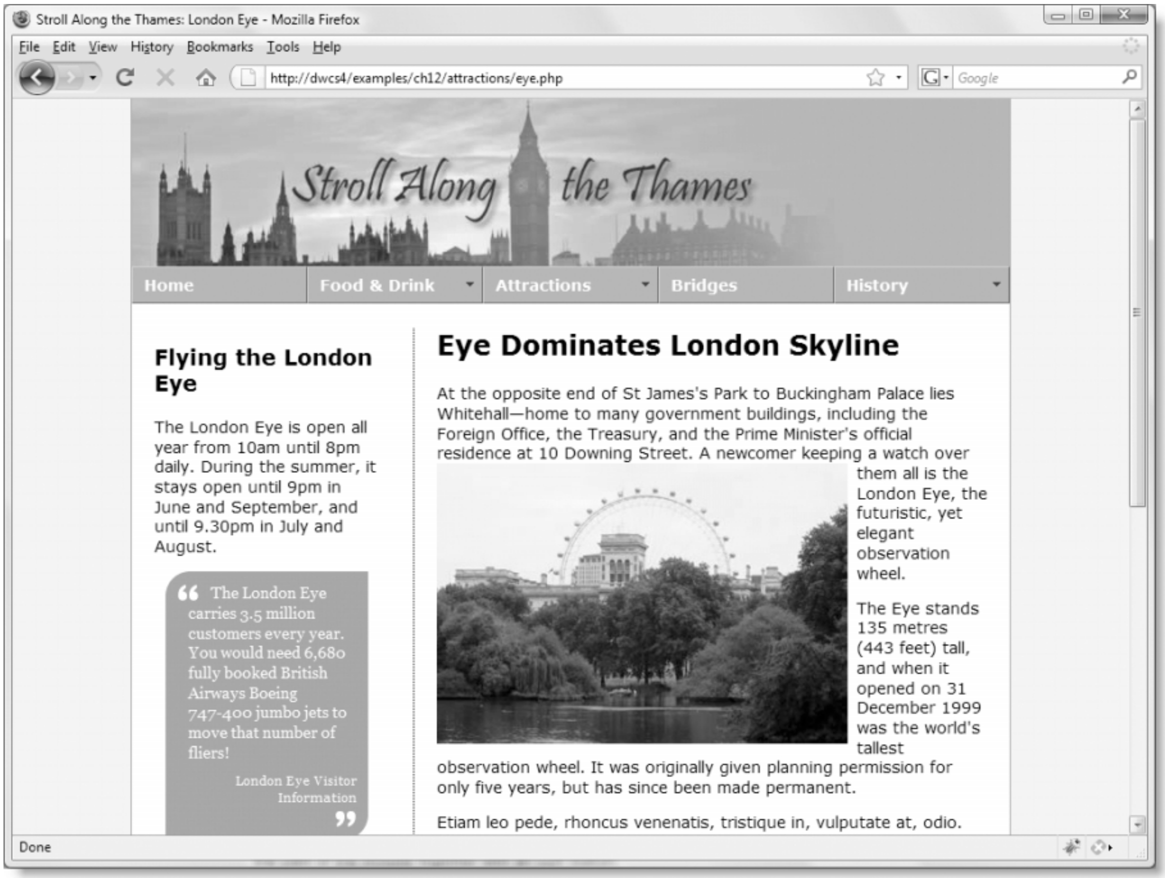
Files of type: All Files (*.*)

URL: ../ch12/stroll_horiz.php

Relative to: Document menu.inc.php

Change default Link Relative To in the site definition.

3. Click OK to update the link. Then select the London Eye link, and create a link to `eye.php`, which is in the `examples/ch12/attractions` folder. Also make sure Relative to is set to Document, and click OK to update the link. There is no need to move `eye.php` to the `workfiles` folder. I have created the page so that it automatically includes your version of `menu.inc.php` from `workfiles/includes`.
4. Save `menu.inc.php`, switch back to `horiz_stroll.php`, and press F12/Opt+F12 to load the page into a browser. Click Yes when asked whether you want to update the page on the testing server.
5. Mouse over Attractions in the menu bar to reveal the drop-down menu, and click London Eye. You should see the page shown in Figure 12-7.


Stroll Along the Thames: London Eye - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://dwcs4/examples/ch12/attractions/eye.php
Google
Stroll Along the Thames
Home
Food & Drink
Attractions
Bridges
History
Done

Flying the London Eye

The London Eye is open all year from 10am until 8pm daily. During the summer, it stays open until 9pm in June and September, and until 9.30pm in July and August.

The London Eye carries 3.5 million customers every year. You would need 6,680 fully booked British Airways Boeing 747-400 jumbo jets to move that number of fliers!

London Eye Visitor Information

Eye Dominates London Skyline

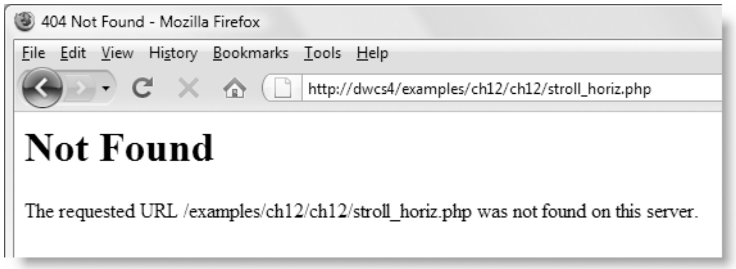
At the opposite end of St James's Park to Buckingham Palace lies Whitehall—home to many government buildings, including the Foreign Office, the Treasury, and the Prime Minister's official residence at 10 Downing Street. A newcomer keeping a watch over them all is the London Eye, the futuristic, yet elegant observation wheel.

The Eye stands 135 metres (443 feet) tall, and when it opened on 31 December 1999 was the world's tallest observation wheel. It was originally given planning permission for only five years, but has since been made permanent.

Etiam leo pede, rhoncus venenatis, tristique in, vulputate at, odio.

Figure 12-7. The menu navigates successfully to a page at a different level in the site hierarchy.

6. Now click the Home link in the new page. This time, you should see something similar to Figure 12-8.


404 Not Found - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://dwcs4/examples/ch12/ch12/stroll_horiz.php

Not Found

The requested URL /examples/ch12/ch12/stroll_horiz.php was not found on this server.

Figure 12-8. Document-relative links prevent the menu from navigating back to the correct location.

If you're using the same site structure as I am, you'll see that the menu has tried to find `stroll_horiz.php` in the examples folder, rather than return to your `workfiles` folder. Not only that, but it's trying to find two levels of `ch12`. Although the browser was able to find the correct file the first time, document-relative links make it impossible for an include file to navigate through a complex site structure. Using document-relative links works only if you keep everything in the same folder.

7. Return to `menu.inc.php`, and update the Home and London Eye links. They should still point to the same pages, but this time select Site Root as the value for Relative to. The values in the URL field should look like this:

```
/workfiles/ch12/stroll_horiz.php
/examples/ch12/attractions/eye.php
```

This works only if you have set up your testing server as a virtual host in Apache or as a web site in IIS7.

If you defined your site in a subfolder of the server root, you need to prefix these values with the name of the subfolder preceded by a forward slash. For example, if you created the site in a subfolder of the server root called `dwcs4` and you use `http://localhost/dwcs4/workfiles/ch12/stroll_horiz.php` to display the first page, you need to manually adjust the preceding values like this:

```
/dwcs4/workfiles/ch12/stroll_horiz.php
/dwcs4/examples/ch12/attractions/eye.php
```

If you have built the site in your `Sites` folder on a Mac, you need to add your username as well, like this:

```
/~username/dwcs4/workfiles/ch12/stroll_horiz.php
/~username/dwcs4/examples/ch12/attractions/eye.php
```

8. The script that initializes the menu bar also uses document-relative links like this:

```
var MenuBar1 = new Spry.Widget.MenuBar("MenuBar1", ➥
{imgDown:"../../SpryAssets/SpryMenuBarDownHover.gif", ➥
imgRight:"../../SpryAssets/SpryMenuBarRightHover.gif"});
```

Because the include file always remains in the same location relative to the SpryAssets folder, you don't need to edit these links. However, for the sake of consistency, it's a good idea to do so. Remove the ../.. from the beginning of both links like this:

```
var MenuBar1 = new Spry.Widget.MenuBar("MenuBar1", ➥
{imgDown:"/SpryAssets/SpryMenuBarDownHover.gif", ➥
imgRight:"/SpryAssets/SpryMenuBarRightHover.gif"});
```

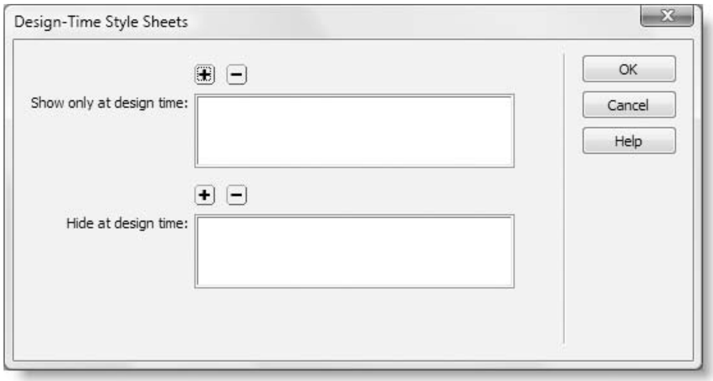
9. Save menu.inc.php and reload stroll_horiz.php in a browser. You should now be able to navigate successfully between the Home and London Eye pages.

The instructions in step 7 for prefixing the links with the server root subfolder are purely for the benefit of this exercise. When deploying the file on a live site, you would need to remove the /dwcs4 *or* /~username/dwcs4 *from each link, making the process cumbersome and prone to error. If you plan to use links in include files, it's essential to set up your testing server in a virtual host or a standalone web site in IIS7 (see Chapter 2 for details).*

Applying styles with design-time style sheets

Although Dreamweaver displays the menu normally in stroll_horiz.php, it looks completely different in menu.inc.php. As Figure 12-6 shows, the menu is completely unstyled; all you can see is the underlying series of nested unordered lists. Design-time style sheets let you apply the styles in an external style sheet to a page or code fragment without the need to attach the style sheet directly to the page. As the name suggests, the style sheet is applied only at design time; in other words, it's applied in Design view.

To apply design-time style sheets to a page or an include file, select CSS Styles ➤ Design-time from either the Format menu or from the context menu when right-clicking in Design view. This opens the Design-Time Style Sheets dialog box, as shown in the following screenshot:



The dialog box has two sections. The first one, Show only at design time, lets you apply a style sheet without attaching it to the file. The second one, Hide at design time, works with style sheets that are attached to a file, letting you hide the effect of selected style sheets while working in Design view. It's particularly useful when working with style sheets for different media, such as print and screen.

Both sections work the same way: add a style sheet to the list by clicking the plus (+) button and navigating to the style sheet in the site file system. The rules of the CSS cascade apply, so add multiple style sheets in the same order as to the original page. To remove a style sheet, highlight it, and click the minus (–) button. Figure 12-9 shows `menu.inc.php` after applying `workfiles/ch12/stroll.css` and `SpryAssets/SpryMenuBarHorizontal_stroll.css` as design-time style sheets. It now looks the same as in the page it was extracted from.

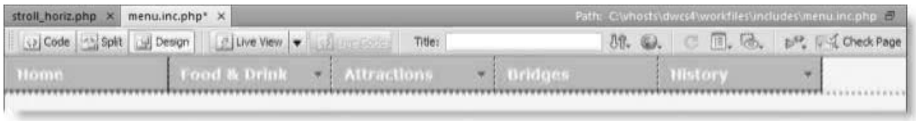


Figure 12-9. After applying design-time style sheets, the include file looks the same as in the original page.

With the design-time style sheets applied, you can manipulate the styles of the include file by changing the class or ID of individual elements. You can also change the style rules in the external style sheets through the CSS Styles panel or the CSS view of the Property inspector. But—and it's a rather large one—you should remember that the code fragment you're working with is no longer in the context of its parent page. As a result, you cannot access the Code Navigator by holding down Alt/Opt+Cmd and clicking in the page, nor are the styles preserved in Live view. More importantly, the full effect of the CSS cascade may not be accurately reflected if particular styles are dependent on being inside a parent element that's not part of the code fragment. Also, changes made to the external style sheet may have unexpected consequences on other parts of your design. Although useful, design-time style sheets have their limitations.

Another drawback is that design-time style sheets can be applied to only one page at a time. There is a commercial extension available that lets you apply design-time style sheets to an entire site. See `http://www.communitymx.com/abstract.cfm?cid=61265` for details. Dreamweaver stores details of style sheets applied to a page in this way in a subfolder called `_notes`. The subfolder is hidden in the Files panel but can be inspected in Windows Explorer or Finder.

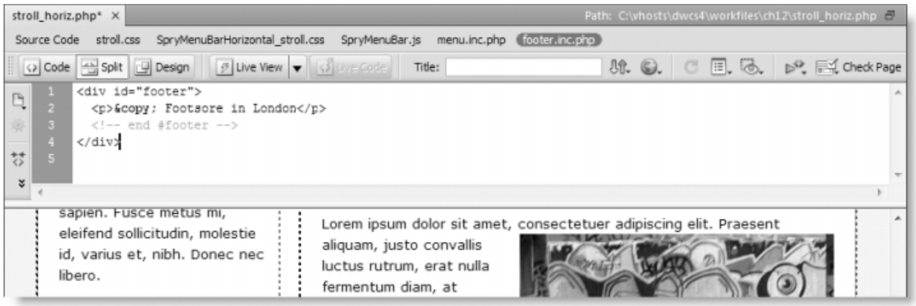
Adding dynamic code to an include

The footer of a page frequently contains details that might change, such as company address or telephone number, making it an ideal candidate for an include file.

Automatically updating a copyright notice

The footer in `stroll_horiz.php` contains only a copyright notice, which normally changes only once a year, but with a little PHP magic, you can get it to update automatically at the stroke of midnight on New Year's Eve every year. Continue working with the files from the previous exercise.

1. Create a PHP page, and save it in `workfiles/includes` as `footer.inc.php`. Switch to Code view, and remove all code so the file is completely blank. Switch to Design view.
2. Open `stroll_horiz.php` in Design view, and click anywhere inside the copyright notice at the bottom of the page. Select the entire footer by clicking <div#footer> in the Tag selector, and cut it to your clipboard.
3. Without moving the insertion point, click the Include button on the PHP tab of the Insert bar. Dreamweaver opens Split view with the cursor placed between the parentheses of an `include()` block. Type a single quote, click the Browse icon as before to insert the path to `footer.inc.php`, and type a closing quote.
4. Switch to `footer.inc.php`, and paste the contents of your clipboard into Design view (do *not* paste into Code view—remember always to paste back to the same view as you cut from). The footer is unstyled, but if you save `footer.inc.php`, switch to `stroll_horiz.php`, and click in Design view, then you'll see the footer properly styled as though you had never moved it.
5. Close the separate tab containing `footer.inc.php`. The rest of the work on the include file needs to be done in Code view, so you can now access it through the Related Files toolbar. Select footer.inc.php in the Related Files toolbar to open it in Split view, as shown in the following screenshot:



```
<div id="footer">
  <p>&copy; Footsore in London</p>
  <!-- end #footer -->
</div>
```

sapien. Fusce metus mi, eleifend sollicitudin, molestie id, varius et, nibh. Donec nec libero.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam, justo convallis luctus rutrum, erat nulla fermentum diam, at

6. A copyright notice should have a year. You could just type it in, but the PHP `date()` function generates the current year automatically. Add the following code like this:

```
<p>&copy;
<?php
ini_set('date.timezone', 'Europe/London');
echo date('Y');
?>
Footsore in London</p>
```

Chapter 17 explains dates in PHP and MySQL in detail, but let's take a quick look at what's happening here. The core part of the code is this line:

```
echo date('Y');
```

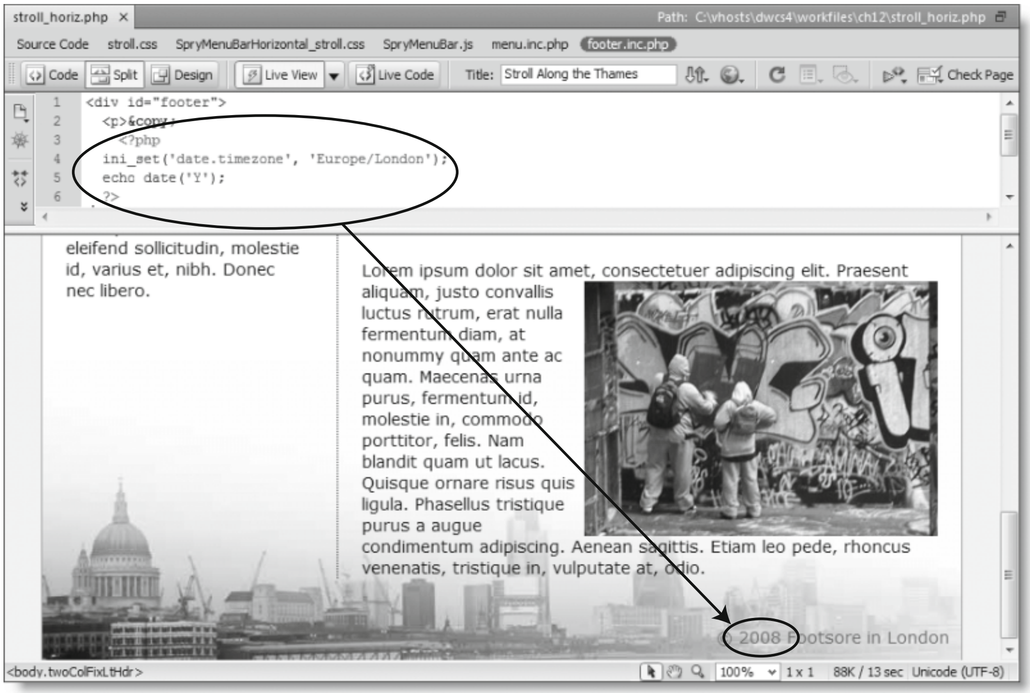
This displays the year using four digits. Make sure you use an uppercase Y. If you use a lowercase y instead, only the final two digits of the year will be displayed.

The reason for the preceding line is because PHP 5.1.0 or higher requires a valid time-zone setting. This should be set in `php.ini`, but if your hosting company forgets to do this, you may end up with ugly error messages in your page.

What if your hosting company is using an earlier version of PHP? No problem. Earlier versions simply ignore this line.

Setting the time zone like this is not only good insurance against error messages, but it also allows you to override the hosting company setting, if your host is in a different time zone from your own. The second argument for `ini_set()` must be one of the time zones listed at `http://docs.php.net/manual/en/timezones.php`.

7. Save both `stroll_horiz.php` and `footer.inc.php`, and click the Live View button. You should see the current year displayed alongside the copyright symbol, as shown in Figure 12-10.



```
<div id="footer">
  <p>&copy;
    <?php
  ini_set('date.timezone', 'Europe/London');
  echo date('Y');
  ?>
```

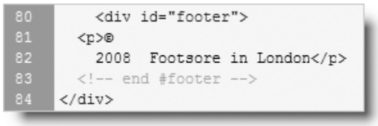
eleifend sollicitudin, molestie id, varius et, nibh. Donec nec libero.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam, justo convallis luctus rutrum, erat nulla fermentum diam, at nonummy quam ante ac quam. Maecenas urna purus, fermentum id, molestie in, commodo porttitor, felis. Nam blandit quam ut lacus. Quisque ornare risus quis ligula. Phasellus tristique purus a augue condimentum adipiscing. Aenean sagittis. Etiam leo pede, rhoncus venenatis, tristique in, vulputate at, odio.

© 2008 Footsore in London

Figure 12-10. The PHP code generates the current year and displays it in the page.

8. Click the Live Code button, and scroll down to the bottom of the Code view section of the Document window. This shows you the actual code being sent to the browser. As you can see in the following screenshot, the PHP code remains on the server, and only the generated output is visible. The line numbers are different because Live Code merges the include files into the main page.



```
    <div id="footer">
  <p>©
    2008  Footsore in London</p>
  <!-- end #footer -->
</div>
```

9. Copyright notices normally cover a range of years, indicating when a site was first launched. To improve the copyright notice, you need to know two things: the start year and the current year. Turn off both Live view and Live Code. Select `footer.inc.php` in the Related Files toolbar, if necessary, and change the PHP code like this:

```
<p>&copy;
<?php
ini_set('date.timezone', 'Europe/London');
$startYear = 2008;
$thisYear = date('Y');
if ($startYear == $thisYear) {
  echo $startYear;
} else {
  echo "{$startYear}-{$thisYear}";
}
?>
Footsore in London</p>
```

This uses simple conditional logic (if you're new to PHP, see "Using comparisons to make decisions" in Chapter 10, and take particular note of the use of two equal signs in the conditional statement). The static value of `$startYear` is compared to the dynamically generated value of `$thisYear`. If both are the same, only the start year is displayed; if they're different, you need to display both with a hyphen between them.

I've used curly braces around the variables in the following line:

```
echo "{$startYear}-{$thisYear}";
```

This is because they're in a double-quoted string that contains no whitespace. The curly braces enable the PHP engine to identify the beginning and end of the variables. Since hyphens aren't permitted in variable names, you could omit the curly braces on this occasion. However, their presence makes the code easier to read.

10. Save `footer.inc.php`, and toggle Live view on again. Assuming you used the current year for `$startYear`, you'll see no difference, so experiment by changing the value of `$startYear` and alternating between uppercase and lowercase y in the `date()` function.

Depending on the value of $startYear and the current date, you should see something like © 2007–2008 if you used an uppercase Y, and © 2007–08 with a lowercase y.

The values of $startYear, $thisYear, and the name of the copyright owner are the only things you need to change, and you have a fully automated copyright notice. You can check your code against footer.inc.php in examples/includes and stroll_horiz_footer.php in examples/ch12.

Using includes to recycle frequently used PHP code

Includes become really useful when you create PHP code that can be used in any site. A simple example is the POST stripslashes snippet you used in the previous chapter. Instead of putting the code directly inside your script, you could put it in an external file and use include() to incorporate it.

Let's take a look at the code again:

```
// remove escape characters from POST array
if (PHP_VERSION < 6 && get_magic_quotes_gpc()) {
  function stripslashes_deep($value) {
    $value = is_array($value) ? array_map('stripslashes_deep', ➥
$value) : stripslashes($value);
    return $value;
  }
  $_POST = array_map('stripslashes_deep', $_POST);
}
```

It contains nothing but PHP code, and the code itself consists of a conditional statement that removes backslashes from the $_POST array if magic quotes are enabled on the server. To use it successfully as an include, you must do the following two things:

- The code in the external file must be surrounded by PHP tags. Although include() and its related commands are part of PHP, the PHP engine treats everything in an include file as plain text or HTML until it encounters an opening PHP tag. The opening tag must be matched by a closing one at or before the end of the include file.
- The code must be included at the point in the script where you want to run it. In this respect, it's the same as the text and HTML includes earlier in the chapter.

PHP can be used in two main ways: as a **procedural** language and as an **object-oriented** one. In a procedural language, everything is usually in the same page, and the code is executed from top to bottom. However, to avoid the need to retype frequently used sections of script, you can package them up as custom-built functions. An object-oriented language takes the concept of functions much further and packages most of the code in libraries called **classes**.

For an in-depth look at object-oriented PHP, see my PHP Object-Oriented Solutions *(friends of ED, ISBN: 978-1-4302-1011-5).*

That's a vast oversimplification, but in both approaches, unless the contents of an external file define functions or classes, the include command must come at the point in the code where you want to run it. The POST stripslashes snippet does include the definition of the `stripslashes_deep()` function, but it's buried inside a conditional statement. So, the snippet itself is a chunk of procedural code that must be included at the point of the script where it's needed.

However, you can convert the snippet into a new function called `nukeMagicQuotes()` like this:

```
<?php
function nukeMagicQuotes() {
  // remove escape characters from POST array
  if (get_magic_quotes_gpc()) {
    function stripslashes_deep($value) {
      $value = is_array($value) ? array_map('stripslashes_deep', ➥
$value): stripslashes($value);
      return $value;
    }
    $_POST = array_map('stripslashes_deep', $_POST);
  }
}
?>
```

If you save this as `nukequotes.inc.php`, you can include the external file at the beginning of your script and run this function at any stage in your script like this (you can see the code in `feedback_nuke.php` in `examples/ch12` and `nukequotes.inc.php` in `examples/includes`):

```
nukeMagicQuotes();
```

The difference of this approach is that the include file initializes the function, but the function doesn't actually run until it's called in the main body of the script. Since this particular piece of code runs only once, there's no immediate advantage of doing it this way. However, let's say you find a way of improving this script; the changes need to be made only in the external file, saving you the effort of hunting through every page where it might have been used. External files can define more than one function, so you can store frequently used functions together. In this respect, includes are the PHP equivalent of linking external JavaScript files or style sheets.

When functions or classes are stored in an external file, the include command must come before you use the functions or classes in your main script.

Although building your own function library is an important use of includes, you shouldn't ignore the opportunity to recycle procedural code.

Adapting the mail processing script as an include

The mail processing script in the previous chapter performs a series of tasks, some of them specific to the feedback form, others more generic in nature. The next section shows you how to adapt the script and make it generic so that it can handle the output of any feedback form. If you glance ahead at the next few pages, you'll see there's a lot of PHP code. Don't despair. Most of the work involves cutting and pasting from one page to another. At the end, you will have an include file that can be used with just about any online form to process the input and send it by email. This considerably reduces the amount of coding that needs to be done in the page that contains the form itself.

Analyzing the script

To make the script reusable, you need to identify what's specific, what's generic, and whether any of the specific tasks can be made generic. Once you have identified the nature of each task, you need to concentrate the generic ones into a single unit that can be exported to an external file.

Table 12-1 lists the tasks in the order they are currently performed and identifies their roles. You can study the code in `feedback_orig.php` in `examples/ch12`.

Table 12-1. Analysis of the mail processing script

Step	Description	Type
1	Check whether form has been submitted.	Specific
2	Remove magic quotes.	Generic
3	Set to address and subject.	Specific
4	List expected and required fields.	Specific
5	Initialize `$missing` array.	Generic
6	Set default values for checkbox group and multiple-choice list.	Specific
7	Filter suspect content.	Generic
8	Process `$_POST` variables and check for missing fields.	Generic
9	Validate email address.	Generic
10	Build the message body.	Specific
11	Create additional headers.	Specific
12	Send email.	Generic

As you can see from Table 12-1, most tasks are generic, but they don't form a single block. However, step 2 can easily be moved after step 6. That leaves just steps 6, 10, and 11 that get in the way. The easy way to deal with step 6 is to initialize the `$missing` array as part of the specific script. After all, it's only one line.

So, what about step 10? This builds the body of the message, which would appear to be something that's always specific to each form. Let's take another look at that part of the script:

```
    // build the message
    $message = "Name: $name\r\n\r\n";
    $message .= "Email: $email\r\n\r\n";
    $message .= "Comments: $comments\r\n\r\n";
    $message .= 'Interests: '.implode(', ', $interests)."\r\n\r\n";
    $message .= "Subscribe: $subscribe\r\n\r\n";
    $message .= "Visited: $visited\r\n\r\n";
    $message .= 'Impressions of London: '.implode(', ', $views);
```

It doesn't take a genius to work out that the message is built using text labels followed by variables with the same name as the label. Since the variable names come from the `name` attributes in the form, all you need is a way of displaying the `name` attributes as well as the values of each input field. That's easily done with PHP. It's also easy to set default values for variables that contain nothing.

That leaves just step 11, the creation of additional headers. With the exception of the return email address, it doesn't matter when you specify the additional headers. They simply need to be passed to the `mail()` function in step 12. So, you can move the creation of most headers to the form-specific section at the beginning of the script. Table 12-2 shows the revised order of tasks.

Table 12-2. The revised mail processing script

Where defined	Step	Description
Main script		
	1	Check whether form has been submitted.
	2	Set to address and subject.
	3	Set form-specific email headers.
	4	List expected and required fields.
	5	Initialize missing array.
	6	Set default values for checkbox group and multiple-choice list.

Where defined	Step	Description
Include file		
	1	Remove magic quotes.
	2	Filter suspect content.
	3	Process $_POST variables, and check for missing fields.
	4	Validate email address.
	5	Build the message body.
	6	Add return email address to headers.
	7	Send email.

Building the message body with a generic script

Loops and arrays take a lot of the hard work out of PHP scripts, although they can be difficult to understand when you're new to PHP. You may prefer just to use the completed script, but if you're interested in the details, take a look at the following code, and I'll explain how it works:

```
// initialize the $message variable
$message = '';
// loop through the $expected array
foreach($expected as $item) {
  // assign the value of the current item to $val
  if (isset(${$item}) && !empty(${$item})) {
    $val = ${$item};
  } else {
    // if it has no value, assign 'Not selected'
    $val = 'Not selected';
  }
  // if an array, expand as comma-separated string
  if (is_array($val)) {
    $val = implode(', ', $val);
  }
  // add label and value to the message body
  $message .= ucfirst($item).": $val\r\n\r\n";
}
```

This replaces the code for step 10 that was listed in the preceding section. It begins by initializing $message as an empty string. Everything else is inside a foreach loop (see "Looping through arrays with foreach" in Chapter 10), which iterates through the $expected array. This array consists of the name attributes of each form field (name, email, and so on).

A foreach loop assigns each element of an array to a temporary variable. In this case, I have used $item. So, the first time the loop runs, $item is name; the next time it's email, and so on. This means you can use $item as the text label for each form field, but before you can do that, you need to know whether the field contains any value. The code that processes the $_POST variables assigns the value of each field to a variable based on its name attribute ($name, $email, and so on). The rather odd-looking ${$item} is what's known as a **variable variable** (the repetition is deliberate, not a misprint). Since the value of $item is name the first time the loop runs, ${$item} refers to $name. On the next pass through the loop, it refers to $email, and so on.

In effect, what happens is that on the first iteration the following conditional statement

```
if (isset(${$item}) && !empty(${$item})) {
  $val = ${$item};
}
```

becomes this:

```
if (isset($name) && !empty($name)) {
  $val = $name;
}
```

If the variable doesn't exist (which would happen if nothing was selected in a checkbox group) or if it doesn't contain a value, the else clause assigns $val the string Not selected.

So, you now have $item, which contains the label for the field, and $val, which contains the field's value.

The next conditional statement uses is_array() to check whether the field value is an array (as in the case of checkboxes or a multiple-choice list). If it is, the values are converted into a comma-separated string by implode().

Finally, the label and field value are added to $message using the combined concatenation operator (.=). The label ($item) is passed to the ucfirst() function, which converts the first character to uppercase. The concatenation operator (.) joins the label to a double-quoted string, which contains a colon followed by the field value ($val) and two pairs of carriage returns and newline characters.

This code handles all types and any number of form fields. All it needs is for the name attributes to make suitable labels and to be added to the $expected array.

Converting feedback.php to use the generic script

The following instructions show you how to adapt feedback.php from the previous chapter so that it can be recycled for use with most forms. If you don't have a copy of the file from the previous chapter, copy feedback_orig.php from examples/ch12 to workfiles/ch12, and save it as feedback.php.

1. Create a new PHP file, and save it as process_mail.inc.php in workfiles/includes. Switch to Code view, and strip out all existing code.
2. Insert the following code:

   ```
   <?php
   if (isset($_SERVER['SCRIPT_NAME']) && strpos($_SERVER['SCRIPT_NAME'],➥
   '.inc.php')) exit;

   ?>
   ```

   This uses the predefined variable $_SERVER['SCRIPT_NAME'] and the strpos() function to check the name of the current script. If it contains .inc.php, that means somebody is trying to access the include file directly through a browser, so the exit command brings the script to a halt. When accessed correctly as an include file, $_SERVER['SCRIPT_NAME'] contains the name of the parent file, so unless you also give that the .inc.php file name extension, the conditional statement returns false and runs the rest of the script as normal.

   Calling process_mail.inc.php directly shouldn't have any negative effect, but if display_errors is enabled on your server, it generates error messages that might be useful to a malicious attacker. This simple security measure prevents the script running unless it's accessed correctly.
3. Cut the POST stripslashes code from the top of feedback.php, and paste it on the blank line before the closing PHP tag in process_mail.inc.php.
4. Leave $to, $subject, $expected, and $required in feedback.php. Cut the remaining PHP code above the DOCTYPE declaration (DTD), except for the closing curly brace and PHP tag. The following code should be left above the DTD in feedback.php:

   ```
   <?php
   if (array_key_exists('send', $_POST)) {
     //mail processing script
     $to = 'me@example.com'; // use your own email address
     $subject = 'Feedback from Essential Guide';

     // list expected fields
     $expected = array('name', 'email', 'comments', 'interests', ➥
   'subscribe', 'visited', 'views');
     // set required fields
     $required = array('name', 'comments', 'interests', 'visited', ➥
   'views');
   ```

12

```
  // create empty array for any missing fields
  $missing = array();
  // set default values for variables that might not exist
  if (!isset($_POST['interests'])) {
    $_POST['interests'] = array();
  }
  if (!isset($_POST['views'])) {
    $_POST['views'] = array();
  }
  // minimum number of required checkboxes
  $minCheckboxes = 2;
  // if fewer than required, add to $missing array
  if (count($_POST['interests']) < $minCheckboxes) {
    $missing[] = 'interests';
  }
}
?>
```

5. Paste into process_mail.inc.php just before the closing PHP tag the code you cut from feedback.php.

6. Cut the following two lines from process_mail.inc.php:

```
// create additional headers
$headers = "From: Essential Guide<feedback@example.com>\r\n";
$headers .= 'Content-Type: text/plain; charset=utf-8';
```

7. Paste them into feedback.php just before the closing curly brace of the code shown in step 4 like this:

```
  if (count($_POST['interests']) < $minCheckboxes) {
    $missing[] = 'interests';
  }
  // create additional headers
  $headers = "From: Essential Guide<feedback@example.com>\r\n";
  $headers .= 'Content-Type: text/plain; charset=utf-8';
}
?>
```

8. Replace the code that builds the message with the generic version shown at the beginning of this section. The full listing for process_mail.inc.php follows, with the new code highlighted in bold:

```
<?php
if (isset($_SERVER['SCRIPT_NAME']) && strpos($_SERVER['SCRIPT_NAME'],➥
'.inc.php')) exit;
// remove escape characters from POST array
if (get_magic_quotes_gpc()) {
  function stripslashes_deep($value) {
    $value = is_array($value) ? array_map('stripslashes_deep', ➥
$value) : stripslashes($value);
```

```
    return $value;
  }
  $_POST = array_map('stripslashes_deep', $_POST);
}

// assume that there is nothing suspect
$suspect = false;
// create a pattern to locate suspect phrases
$pattern = '/Content-Type:|Bcc:|Cc:/i';

// function to check for suspect phrases
function isSuspect($val, $pattern, &$suspect) {
  // if the variable is an array, loop through each element
  // and pass it recursively back to the same function
  if (is_array($val)) {
    foreach ($val as $item) {
      isSuspect($item, $pattern, $suspect);
    }
  } else {
    // if one of the suspect phrases is found, set Boolean to true
    if (preg_match($pattern, $val)) {
      $suspect = true;
    }
  }
}

// check the $_POST array and any subarrays for suspect content
isSuspect($_POST, $pattern, $suspect);

if ($suspect) {
  $mailSent = false;
  unset($missing);
} else {
  // process the $_POST variables
  foreach ($_POST as $key => $value) {
    //assign to temporary variable and strip whitespace if not an array
    $temp = is_array($value) ? $value : trim($value);
    // if empty and required, add to $missing array
    if (empty($temp) && in_array($key, $required)) {
      array_push($missing, $key);
    } elseif (in_array($key, $expected)) {
      // otherwise, assign to a variable of the same name as $key
      ${$key} = $temp;
    }
  }
}
```

```
// validate the email address
if (!empty($email)) {
  // regex to identify illegal characters in email address
  $checkEmail = '/^[^@]+@[^\s\r\n\'";,@%]+$/';
  // reject the email address if it deosn't match
  if (!preg_match($checkEmail, $email)) {
    $suspect = true;
    $mailSent = false;
    unset($missing);
  }
}

// go ahead only if not suspsect and all required fields OK
if (!$suspect && empty($missing)) {
  // initialize the $message variable
  $message = '';
  // loop through the $expected array
  foreach($expected as $item) {
    // assign the value of the current item to $val
    if (isset(${$item}) && !empty(${$item})) {
      $val = ${$item};
    } else {
      // if it has no value, assign 'Not selected'
      $val = 'Not selected';
    }
    // if an array, expand as comma-separated string
    if (is_array($val)) {
      $val = implode(', ', $val);
    }
    // add label and value to the message body
    $message .= ucfirst($item).": $val\r\n\r\n";
  }

  // limit line length to 70 characters
  $message = wordwrap($message, 70);

  // create Reply-To header
  if (!empty($email)) {
    $headers .= "\r\nReply-To: $email";
    }

  // send it
  $mailSent = mail($to, $subject, $message, $headers);
  if ($mailSent) {
    // $missing is no longer needed if the email is sent, so unset it
    unset($missing);
    }
  }
?>
```

9. All that remains is to include the mail processing script. Since the form won't work without it, it's a wise precaution to check that the file exists and is readable before attempting to include it. The following is a complete listing of the amended code above the DOCTYPE declaration in feedback.php. The new code, including the $header pasted in the previous step, is highlighted in bold.

```
<?php
if (array_key_exists('send', $_POST)) {
  //mail processing script
  $to = 'me@example.com'; // use your own email address
  $subject = 'Feedback from Essential Guide';

  // list expected fields
  $expected = array('name', 'email', 'comments', 'interests', ➥
'subscribe', 'visited', 'views');
  // set required fields
  $required = array('name', 'comments', 'interests', 'visited', ➥
'views');
  // create empty array for any missing fields
  $missing = array();
  // set default values for variables that might not exist
  if (!isset($_POST['interests'])) {
    $_POST['interests'] = array();
  }
  if (!isset($_POST['views'])) {
    $_POST['views'] = array();
  }
  // minimum number of required checkboxes
  $minCheckboxes = 2;
  // if fewer than required, add to $missing array
  if (count($_POST['interests']) < $minCheckboxes) {
    $missing[] = 'interests';
  }
  $headers = "From: Essential Guide<feedback@example.com>\r\n";
  $headers .= 'Content-Type: text/plain; charset=utf-8';
  $process = '../includes/process_mail.inc.php';
  if (file_exists($process) && is_readable($process)) {
    include($process);
  } else {
    $mailSent = false;
  }
}
?>
```

The path to process_mail.inc.php is stored in $process. This avoids the need to type it three times. The conditional statement uses two functions with self-explanatory names: file_exists() and is_readable(). If the file is OK, it's included. If not, $mailSent is set to false. This displays the warning that there was a problem sending the message.

10. To be super-efficient, send yourself an email alerting you to the problem with the include file by amending the conditional statement like this:

```
if (file_exists($process) && is_readable($process)) {
  include($process);
} else {
  $mailSent = false;
  mail($to, 'Server problem', "$process cannot be read", $headers);
}
```

You can check the final code in `feedback_process.php` in `examples/ch12` and `process_mail.inc.php` in `examples/includes`.

Because `process_mail.inc.php` uses generic variables, you can slot this include file into any page that processes a form and sends the results by email. The only proviso is that you must use the same variables as in step 9, namely, `$to`, `$subject`, `$expected`, `$required`, `$missing`, `$minCheckboxes`, `$headers`, and `$mailSent`. If you don't want to set a minimum number of checkboxes, set `$minCheckboxes` to 0.

Programming purists would criticize this use of procedural code, arguing that a more robust solution should be built with object-oriented code. An object-oriented solution would be better. In fact, I have created one in my book, *PHP Object-Oriented Solutions*, but it would be more difficult for a PHP beginner to adapt. It also requires a minimum of PHP 5.2. The purpose of this exercise has been to demonstrate how even procedural code can be recycled with relatively little effort. It also prepares the ground for customizing the PHP code automatically generated by Dreamweaver. With the exception of the XSL Transformations server behavior (covered in Chapter 18), Dreamweaver uses procedural code.

Avoiding the "headers already sent" error

A problem that you're bound to encounter sooner or later is this mysterious error message:

```
Warning: Cannot add header information - headers already sent
```

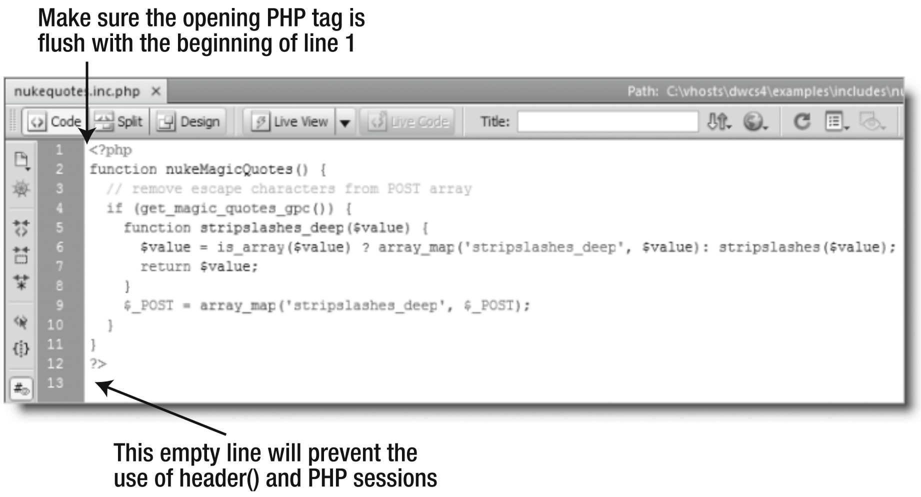
It happens when you use `header()` to redirect a page, as described in the previous chapter, or with PHP sessions (covered in Chapter 15). Frequently, the cause of the problem lies in an include file. The other main culprit lurks inside the main file just before you include the external file.

Using `header()` or starting a PHP session must be done before any output is sent to the browser. This includes not only HTML but also any whitespace. As far as PHP is concerned, *whitespace means any space, tab, carriage return, or newline character* outside a PHP block. Why the error message is so mysterious—and causes so much head banging—is because the whitespace is often at the end of an include file. Use the line numbers in Code view, as shown in Figure 12-11, to make sure there are no blank lines at the end of an include file. Also make sure that there is no whitespace after the closing PHP tag on the final line.

Whitespace *inside* the PHP tags is unimportant, but the PHP code must not generate any HTML output before using `header()` or starting a session. The same applies to the parent page: there must be no whitespace before the opening PHP tag.

On rare occasions, the error is triggered by an invisible control character at the beginning of the file. Use View ➤ Code View Options ➤ Hidden Characters to check, and delete the character.

Since Dreamweaver CS3 adopted UTF-8 as its default encoding, an increasing number of people have reported problems with headers being already sent, even if they've removed all of the whitespace as specified earlier. The reason is because they have selected Include Unicode Signature (BOM) *in the* New Document *category of* Preferences *(*Edit ➤ Preferences*, or* Dreamweaver ➤ Preferences *on a Mac) or in the* Title/Encoding *category of* Page Properties *(see Chapter 4). BOM stands for* byte-order mark*, which is used by some versions of Unicode to indicate how the data is stored. PHP interprets a BOM as output, preventing the use of* `header()` *or sessions. UTF-8 does not require a BOM. Make sure the option to include it is deselected in all PHP pages (this is the Dreamweaver default setting).*


Make sure the opening PHP tag is flush with the beginning of line 1
nukequotes.inc.php
Path: C:\vhosts\dwcs4\examples\includes\n
Code
Split
Design
Live View
Live Code
Title:
This empty line will prevent the use of header() and PHP sessions

```
<?php
function nukeMagicQuotes() {
  // remove escape characters from POST array
  if (get_magic_quotes_gpc()) {
    function stripslashes_deep($value) {
      $value = is_array($value) ? array_map('stripslashes_deep', $value): stripslashes($value);
      return $value;
    }
    $_POST = array_map('stripslashes_deep', $_POST);
  }
}
?>

```

Figure 12-11. Eliminate whitespace outside the PHP tags to avoid the "headers already sent" error.

Chapter review

This chapter has given you a thorough overview of PHP includes, their advantages, and their pitfalls. Once you understand the potential pitfalls, includes are very easy to use. The PHP code generated by Dreamweaver uses them all the time, so at a minimum you need to know how to deal with the "headers already sent" error even if you don't yet have the confidence to start creating your own include files.

When you first start working with PHP, the idea of splitting a page into its various component parts can be a difficult concept to come to terms with, particularly if you come from a nonprogramming background. So, in the next chapter, we're going to take a brief respite from PHP coding to look at Dreamweaver templates and a new feature called Adobe InContext Editing. Templates are a way of building a master page that contains all the common elements for a site, spawning child pages from the master, and updating all the child pages automatically whenever changes are made to the master. InContext Editing bears many similarities to templates but is an online hosted service that lets authorized users update the content in certain parts of a page.